МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ «НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ НИЖЕГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ им. Н. И. ЛОБАЧЕВСКОГО»

ИНСТИТУТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МАТЕМАТИКИ И МЕХАНИКИ

Отчёт по лабораторной работе №6

**Классы для работы с векторами и матрицами**

Выполнила:

студентка ф-та ИТММ ПМИ – 381903-3

Семибабнова Анна Владимировна

Проверил:

ассистент кафедры МОСТ

Лебедев Илья Геннадьевич

Нижний Новгород

2020 г.

Содержание

[Введение 3](#_Toc24628114)

[Постановка задачи 4](#_Toc24628115)

[Руководство пользователя 5](#_Toc24628116)

[Руководство программиста 6](#_Toc24628117)-11

[Эксперименты 11-14](#_Toc24628118)

[Заключение 15](#_Toc24628119)

[Литература 16](#_Toc24628120)

[Приложение 17-26](#_Toc24628121)

# 1.Введение

Число является одним из основных понятий математики. Понятие числа развивалось в тесной связи с изучением величин; эта связь сохраняется и теперь. Во всех разделах современной математики и информатики приходится рассматривать разные величины и пользоваться числами.

В этой лабораторной работе мы столкнемся с такими понятиями, как вектор и матрица.

**Вектор** - направленный отрезок прямой, то есть отрезок, для которого указано, какая из его граничных точек является началом, а какая — концом. **Матрица** - [математический объект](https://ru.wikipedia.org/wiki/%D0%9C%D0%B0%D1%82%D0%B5%D0%BC%D0%B0%D1%82%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D0%B9_%D0%BE%D0%B1%D1%8A%D0%B5%D0%BA%D1%82), записываемый в виде прямоугольной таблицы элементов [кольца](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BB%D1%8C%D1%86%D0%BE_(%D0%BC%D0%B0%D1%82%D0%B5%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0)) или [поля](https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D0%BB%D0%B5_(%D0%BC%D0%B0%D1%82%D0%B5%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0)) (например, [целых](https://ru.wikipedia.org/wiki/%D0%A6%D0%B5%D0%BB%D0%BE%D0%B5_%D1%87%D0%B8%D1%81%D0%BB%D0%BE), [действительных](https://ru.wikipedia.org/wiki/%D0%94%D0%B5%D0%B9%D1%81%D1%82%D0%B2%D0%B8%D1%82%D0%B5%D0%BB%D1%8C%D0%BD%D0%BE%D0%B5_%D1%87%D0%B8%D1%81%D0%BB%D0%BE) или [комплексных](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D0%BB%D0%B5%D0%BA%D1%81%D0%BD%D0%BE%D0%B5_%D1%87%D0%B8%D1%81%D0%BB%D0%BE) чисел), которая представляет собой совокупность [строк](https://ru.wikipedia.org/w/index.php?title=%D0%A1%D1%82%D1%80%D0%BE%D0%BA%D0%B0_%D0%BC%D0%B0%D1%82%D1%80%D0%B8%D1%86%D1%8B&action=edit&redlink=1) и [столбцов](https://ru.wikipedia.org/w/index.php?title=%D0%A1%D1%82%D0%BE%D0%BB%D0%B1%D0%B5%D1%86_%D0%BC%D0%B0%D1%82%D1%80%D0%B8%D1%86%D1%8B&action=edit&redlink=1), на пересечении которых находятся её элементы. Количество строк и столбцов задает размер матрицы.

Для матрицы определены следующие [алгебраические](https://ru.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%B5%D0%B1%D1%80%D0%B0) операции:

* [сложение](https://ru.wikipedia.org/wiki/%D0%A1%D0%BB%D0%BE%D0%B6%D0%B5%D0%BD%D0%B8%D0%B5) матриц, имеющих один и тот же размер;
* [умножение матриц](https://ru.wikipedia.org/wiki/%D0%A3%D0%BC%D0%BD%D0%BE%D0%B6%D0%B5%D0%BD%D0%B8%D0%B5_%D0%BC%D0%B0%D1%82%D1%80%D0%B8%D1%86) подходящего размера (матрицу, имеющую n {\displaystyle n} столбцов, можно умножить справа на матрицу, имеющую n {\displaystyle n} n строк);
* в том числе умножение на матрицу вектора (по обычному правилу матричного умножения; вектор является в этом смысле частным случаем матрицы);
* умножение матрицы на элемент основного кольца или поля (то есть [скаляр](https://ru.wikipedia.org/wiki/%D0%A1%D0%BA%D0%B0%D0%BB%D1%8F%D1%80)).

Для того чтобы работать с векторами и матрицами в программировании необходимы классы. **Класс** — это тип структуры, позволяющий включать в описание типа не только элементы данных, но и функции (функции-элементы или методы).

Также нам понадобятся шаблоны функций и шаблоны классов**. Шаблоны функций** – это обобщенное описание поведения функций, которые могут вызываться для объектов разных типов. **Шаблоны классов** – обобщенное описание пользовательского типа, в котором могут быть параметризованы атрибуты и операции типа.

# 2.Постановка задачи

1. Написать классы для работы с векторами и матрицами использовать шаблоны.

2. Продемонстрировать их работу на примере (написать в main пример).

Должны быть:

* конструкторы (по умолчанию, инициализатор, копирования).
* деструктор.
* доступ к защищенным полям.
* перегруженные операции: +, -, \*, /, =, ==, [].
* потоковый ввод и вывод.
* перегруженные операции +, -, \*, / должны быть реализованы для векторов (вектор +, -, \*, / вектор), матриц (матрица +, -, \* матрица), матрично-векторные (матрица \* вектор и наоборот).
* в классе вектор должна быть возможность отсортировать его тремя способами.

3. Сравнить время работы, и сделать выводы.

# 3.Руководство пользователя

Данная программа написана с помощью Microsoft Visual Studio 2019 на языке C++.

Как пользоваться:

1. Запустить консоль программы «lab\_vector.exe»
2. Создать объект типа vector или matrix
3. Произвести все необходимые операции над вектором или матрицей.

# 

# 4.Руководство программиста

***Описание структуры программы***

Программа состоит из одного решения, которое называется «lab\_vector».

В решении содержится 5 элементов: «vector.h», «vector\_procedure.h», «matrix.h», «matrix\_procedure.h», «main.cpp».

В «vector.h» определен класс **vector.**

В «vector\_procedure» вынесены все определения методов класса vector.

В «matrix.h» определен класс **matrix**.

В «matrix\_procedure.h» вынесены все определения методов класса matrix.

В «main.cpp» определена стандартная функция int main.

***Описание структуры программы***

В программе определены шаблонные классы **Class Matrix** и **Class Vector**.

Внутри класса matrix определены следующие поля:

• Type\*\* matrix – шаблонный двойной указатель

• int width – целочисленный размер ширины матрицы

• int length – целочисленный размер длины матрицы

Внутри класса matrix определен следующий набор **public-методов**:

• Matrix() – конструктор по умолчанию, не принимает никаких параметров, инициализирует все три поля 0 через списки инициализации

• Matrix(int, int) – конструктор инциализатор, принимает на вход два параметра типа int, создает динамический массив заданных размеров

• Matrix(const Matrix& lhs) – конструктор копирования, принимает на вход объект типа matrix, создает объект с теми же характеристиками, что и переданный, и матрицей того же содержимого

• Virtual ~Matrix() – виртуальный деструктор

• int get\_width() – метод, возвращающий ширину матрицы

• int get\_length() – метод, возвращающий длину матрицы

• Type\*\* get\_array() – метод, возвращающий массив, содержащий в себе все элементы матрицы

• void set\_width\_length(int,int) – метод, принимающий новые размеры матрицы, приводящий матрицу к этим размерам

• friend Matrix operator +(Matrix & lhs, Matrix & rhs) - перегрузка оператора суммы

• friend Matrix operator –(Matrix& lhs, Matrix& rhs) - перегрузка оператора ­разности

• friend Matrix operator \*(const Matrix& lhs, const Matrix& rhs) - перегрузка оператора умножения

• Matrix& operator =(const Matrix & lhs) - перегрузка оператора равенства

• friend bool operator ==(Matrix& lhs, Matrix& rhs) - перегрузка оператора сравнения

• friend ostream& operator <<(ostream& out, const Matrix& rhs) - перегрузка оператора вывода, позволяет выводить матрицы на экран

• friend istream& operator >>(istream& in, Matrix& rhs) - перегрузка оператора ввода

**Privat-метод**:

• void MemorySize(int, int) - выделяет память до размеров матрицы, переданных в функцию;

Шаблонный класс vector является public, наследником класса matrix, поэтому никаких полей в этом классе не определено. Почти все методы, он наследует от matrix, поэтому внутри него, определены только специфичные для него методы:

• T1\* get\_vector() - метод, возвращающий массив, содержащий в себе все элементы объекта vector

• T1 operator [](const int) - перегрузка оператора индексации, позволяет возвращать элемент вектора по индексу

• void set\_length(int) - метод, принимающий новые размеры вектора, приводящий вектор к этим размерам

• friend Type operator \*(Vector<Type>& lhs, Vector & rhs) - перегрузка оператора умножить, возвращает скалярное произведение операндов

• friend Vector operator /(Vector& lhs, Vector& rhs) – перегрузка оператора / возвращает вектор, координаты которого являются частным от деления двух соответствующих координат операндов

• friend istream& operator >>(istream& in, Vector& rhs) - перегрузка оператора ввода

• clock\_t BubbleSort() - сортировка «Пузырек».

• clock\_t InsertionSort() - сортировка «Вставка».

• clock\_t QuickSort() - сортировка «Быстрая».

***Описание алгоритмов***

1. **Сортировка пузырьком**

**Изображение выглядит как электроника, черный

Автоматически созданное описание**Главной идеей алгоритма является то, что сравниваются два соседних элемента, и если они стоят в неправильном порядке, то эти элементы меняются местами. Таким образом, за каждый проход в конце массива встает наибольшее число, если сортируется по возрастанию, или наименьшее, если по убыванию. Благодаря этому, программа может не проверят его на следующей итерации.

Рис.1. Алгоритм сортировки пузырьком

1. **Сортировка вставкой**

Изображение выглядит как снимок экрана

Автоматически созданное описаниеВ начале сортировки первый элемент массива считается отсортированным, все остальные — не отсортированные. На каждом шаге сортировки сравнивается текущий элемент со всеми элементами в отсортированной части. Алгоритм вставляет неотсортированный элемент массива в нужную позицию в отсортированной части массива.

Рис.2. Алгоритм сортировки вставкой

1. **Быстрая сортировка**

Выбирается опорный элемент, который разбивает массив примерно на две равные части. Дальше массив сортируется таким образом, что все элементы, которые меньше опорного элемента, встают слева, а которые больше, справа. Дальше выбирается снова опорный элемент для подмассива, который оказался слева от первоначального опорного элемента и то же самое для правого. С помощью рекурсивной сортировки каждой части массива, программа выдает полностью отсортированный массив.

**Изображение выглядит как текст

Автоматически созданное описание**

Рис.3. Алгоритм быстрой сортировки

Блок-схемы алгоритмов операций над матрицами:

1. Алгоритм суммирования или вычитания матриц:

return A

A[i][j]=Ihs+-rhs

for (int=0;i<length();i++)

for (int=0;i<width();i++)

Создание матрицы с определенными размерами

нет

да

Несовпадение размеров полей матриц

Рис.4. Блок-схема алгоритма суммирования и вычитания матриц

1. Алгоритм деления двух векторов:

нет

да

Несовпадение длин векторов

Vector A(lhs)

Rhs[i]!=0

for (int=0;i<rhs.get\_length();i++)

A.get\_vector()[i]=A.get\_vector/rhs[i]

return A

Рис.5. Блок-схема алгоритма деления двух векторов

1. Алгоритм умножения двух векторов:

нет

да

Несовпадение длин векторов

auto s = 0

for (int=0;i<rhs.get\_length();i++)

s=s+lhs.get\_vector()[i]\*lhs.get\_vector

return s

Рис.6. Блок-схема алгоритма умножения двух векторов

# 5.Эксперименты

Оценим время, которое занимают матричные и векторно-матричные операции, рассмотрим код, отвечающий за суммирование матриц:

Matrix<Type> A(lhs.get\_width(), rhs.get\_length());

for (int i = 0; i < rhs.get\_width(); i++)

{

for (int j = 0; j < rhs.get\_length(); j++)

{

A[i][j] = lhs[i][j] + rhs[i][j];

}

}

Теперь произведем замеры времени сложения квадратных матриц:



Рис.7. Замеры времени, полученные при выполнении сложения матриц (сек.).

Посмотрим на соотношения размеров матриц и времени их суммирования:

1. 1000 элементов за 0.265 сек
2. 2000 элементов за 0.785 сек
3. 4000 элементов за 3.89 сек

**Заметим, что при увеличении количества элементов в 2 раза относительно предыдущего, время увеличивается примерно 4 раза.**

Теперь аналогично оценим асимптотическую сложность матричного умножения:

Matrix<F> C(lhs.get\_width(), rhs.get\_length());

F S;

for (int i = 0; i < lhs.get\_width(); i++)

{

for (int j = 0; j < rhs.get\_length(); j++)

{

S = 0;

for (int k = 0; k < rhs.get\_width(); k++)

{

S = S + lhs.get\_array()[i][k] \* rhs.get\_array()[k][j];

}

C[i][j] = S;

}

}

Теперь произведем замеры времени умножения квадратных матриц (чтобы точнее соотнести результаты замеров с асимптотической сложностью):



Рис.8.Замеры времени, полученные при выполнении умножения матриц (сек.).

Посмотрим на соотношения размеров матриц и времени их умножения:

1. 1000 элементов за 7.559 сек
2. 2000 элементов за 124.24 сек
3. 4000 элементов за 1224.47 сек

Теперь аналогично оценим асимптотическую сложность векторно-матричного умножения:

Matrix<F> C(lhs.get\_width(), rhs.get\_length());

F S;

for (int i = 0; i < lhs.get\_width(); i++)

{

for (int j = 0; j < rhs.get\_length(); j++)

{

S = 0;

for (int k = 0; k < rhs.get\_width(); k++)

{

S = S + lhs.get\_array()[i][k] \* rhs.get\_array()[k][j];

}

C[i][j] = S;

}

}

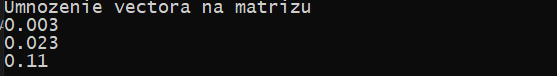
Теперь произведем замеры времени умножения вектора на матрицу:

Рис.9.Замеры времени, полученные при выполнении умножения вектора на матрицу(сек.).

Посмотрим на соотношения размеров матриц и времени умножения:

1. 1000 элементов за 0.003 сек
2. 2000 элементов за 0.023 сек
3. 4000 элементов за 0.11 сек

Для стандартных алгоритмов сортировки (BubbleSort, IsertionSort и QuickSort) асимптотическая сложность уже определена.

Произведем замеры:



Рис.10.Замеры времени, полученные при выполнении сортировки вектора (сек.)



Рис.11.Замеры времени, полученные при выполнении сортировки вектора(сек.)



Рис.12.Замеры времени, полученные при выполнении сортировки вектора(сек.)

Можно заметить, что затрачиваемое на реализацию операции время, растет эквивалентно асимптотической функции.

# 6.Заключение

Таким образом, в ходе лабораторной работы была создана программа, которая продемонстрировала работу с классами векторов и матриц с использованием шаблонов.

Эксперименты показали, что время, затраченное на реализацию матричных, матрично-векторных операций и различных сортировок с некоторыми погрешностями совпадает с асимптотической сложности времени, которого мы ожидали.

# 

# 7.Литература

1. С. Н. Марков. Курс истории математики: Учебное пособие. - Иркутск: Издательство иркутского университета, 1995. 248 с.
2. <http://www.math24.ru/множества-чисел.html>
3. Павловская Т.А. C/C++, Программирование на языке высокого уровня, 2003
4. <https://ru.wikipedia.org>

# 8.Приложение

***Приложение 1. Код программы***

**«vector.h»**

#include "matrix.h"

template <class T1>

class Vector :public Matrix<T1>

{

public:

Vector() :Matrix() {}

Vector(int \_size) :Matrix(1, \_size) {}

Vector(const Vector& t) :Matrix(t) {}

~Vector() {}

T1\* get\_vector();

using Matrix:: operator =;

T1& operator [](const int i);

void set\_length(int \_resize);

template<class J>

friend Vector<J> operator /(Vector<J>& lhs, Vector<J>& rhs);

template<class Y>

friend Y operator \*(Vector<Y>& lhs, Vector<Y>& rhs);

template<class K>

friend istream& operator >> (istream& in, Vector<K>& rhs);

friend void In\_Vector(Vector& lhs)

{

for (int i = 0; i < lhs.get\_length(); i++)

lhs[i] = rand();

}

template<class P>

friend clock\_t BubbleSort(Vector<P>& B);

template<class Q>

friend clock\_t InsertionSort(Vector<Q>& b);

template<class K>

friend clock\_t QuickSort(Vector<K>& p, int first, int last);

};

#include "vector\_proceed.h"

**«vector\_proceed.h»**

template <class T1>

T1\* Vector<T1>::get\_vector()

{

return get\_array()[0];

}

template<class T2>

T2& Vector<T2>::operator [](const int i)

{

return get\_vector()[i];

}

template <class T1>

void Vector<T1>::set\_length(int \_resize)

{

set\_width\_length(1, \_resize);

}

template<class F>

Vector<F> operator /(Vector<F>& lhs, Vector<F>& rhs)

{

if (lhs.get\_length() != rhs.get\_length())throw length\_error("Vector's size not equal");

Vector<F> A(lhs);

for (int i = 0; i < A.get\_length(); i++)

{

if (rhs[i] != 0)

{

A.get\_vector()[i] = A.get\_vector()[i] / rhs[i];

}

}

return A;

}

template <class Y>

Y operator \*(Vector<Y>& lhs, Vector<Y>& rhs)

{

if (lhs.get\_length() != rhs.get\_length())throw length\_error("Vector's size not equal");

Y s = 0;

for (int i = 0; i < rhs.get\_length(); i++)s = s + lhs.get\_vector()[i] \* rhs.get\_vector()[i];

return s;

}

template<class U>

istream& operator >> (istream& in, Vector<U>& rhs)

{

if (rhs.get\_length() == 0)

{

int \_size;

in >> \_size;

rhs.set\_length(\_size);

}

for (int i = 0; i < rhs.get\_length(); i++)

{

in >> rhs.get\_vector()[i];

}

return in;

}

template<class P>

clock\_t BubbleSort(Vector<P>& A)

{

clock\_t t1, t2;

t1 = clock();

P tmp = 0;

int i, j = 0;

t1 = clock();

for (int i = 0; i < A.get\_length(); i++)

{

for (int j = 0; j < i; j++)

{

if (A.get\_vector()[j] > A.get\_vector()[j + 1])

{

tmp = A.get\_vector()[j];

A.get\_vector()[j] = A.get\_vector()[j + 1];

A.get\_vector()[j + 1] = tmp;

}

}

}

t2 = clock();

return (t2 - t1);

}

template<class Q>

clock\_t InsertionSort(Vector<Q>& a)

{

clock\_t t1, t2;

t1 = clock();

Q newElement;

int location;

for (int i = 1; i < a.get\_length(); i++)

{

newElement = .get\_vector()[i];

location = i - 1;

while (location >= 0 && a.get\_vector()[location] > newElement)

{

a.get\_vector()[location + 1] = a.get\_vector()[location];

location = location - 1;

}

a.get\_vector()[location + 1] = newElement;

}

t2 = clock();

return t2 - t1;

}

template<class K>

clock\_t QuickSort(Vector<K>& p, int first, int last)

{

clock\_t t1, t2, t3;

t1 = clock();

int i = first, j = last;

K tmp;

K x = p.get\_vector()[(first + last) / 2];

do

{

while (p.get\_vector()[i] < x)

i++;

while (p.get\_vector()[j] > x)

j--;

if (i <= j)

{

if (i < j)

{

tmp = p.get\_vector()[i];

p.get\_vector()[i] = p.get\_vector()[j];

p.get\_vector()[j] = tmp;

}

i++;

j--;

}

}

while (i <= j);

if (i < last)

t3 = QuickSort(p, i, last);

if (first < j)

t3 = QuickSort(p, first, j);

t2 = clock();

return (t2 - t1);

}

**«matrix.h»**

#include <iostream>

using namespace std;

template <class Type>

class Matrix

{

public:

Matrix() :width(0), length(0), matrix(0) {}

Matrix(int \_a, int \_b);

Matrix(const Matrix& lhs);

virtual ~Matrix();

int get\_width()const;

int get\_length()const;

Type\*\* get\_array();

void set\_width\_length(int \_width, int \_length);

Type\*& operator [](const int i);

template<class V>

friend Matrix<V> operator +(Matrix<V>& lhs, Matrix<V>& rhs);

template <class A>

friend Matrix<A> operator -(Matrix<A>& lhs, Matrix<A>& rhs);

Matrix& operator =(Matrix& lhs);

template <class C>

friend bool operator ==(Matrix<C>& lhs, Matrix<C>& rhs);

template <class D>

friend Matrix<D> operator \*(Matrix<D>& lhs, Matrix<D>& rhs);

template <class E>

friend ostream& operator <<(ostream& out, Matrix<E>& rhs);

template<class W>

friend istream& operator >> (istream& in, Matrix<W>& rhs);

private:

void MemorySize(int \_width, int \_length)

{

for (int i = 0; i < \_width; i++) matrix[i] = new Type[\_length];

}

Type\*\* matrix;

int width;

int length;

};

#include "matrix\_proceed.h"

**«matrix\_proceed.h»**

template<class A1>

Matrix<A1>::Matrix(int \_a, int \_b)

{

if ((\_a < 0) || (\_b < 0))

{

if (\_a < 0) throw length\_error("uncorrect width");

if (\_b < 0) throw length\_error("uncorrect length");

}

matrix = new A1 \* [\_a];

MemorySize(\_a, \_b);

for (int i = 0; i < \_a; i++)

{

for (int j = 0; j < \_b; j++)

{

matrix[i][j] = 0;

}

}

width = \_a;

length = \_b;

}

template <class A2>

Matrix<A2>::Matrix(const Matrix& lhs)

{

matrix = new A2 \* [lhs.width];

MemorySize(lhs.width, lhs.length);

for (int i = 0; i < lhs.width; i++)

{

for (int j = 0; j < lhs.length; j++)

{

matrix[i][j] = lhs.matrix[i][j];

}

}

width = lhs.width;

length = lhs.length;

}

template <class A3>

Matrix<A3>::~Matrix()

{

for (int i = 0; i < width; i++) delete[] matrix[i];

delete[] matrix;

}

template<class A1>

int Matrix<A1>::get\_width() const

{

return width;

}

template <class A2>

int Matrix<A2>::get\_length() const

{

return length;

}

template <class A3>

A3\*\* Matrix<A3>::get\_array()

{

return matrix;

}

template<class A1>

void Matrix<A1>::set\_width\_length(int \_width, int \_length)

{

if ((\_width < 0) || (\_length < 0))

{

if (\_width < 0) throw length\_error("uncorrect width");

if (\_length < 0) throw length\_error("uncorrect length");

}

Matrix<A1>D(width, length);

for (int i = 0; i < width; i++)

{

for (int j = 0; j < length; j++)

{

D.get\_array()[i][j] = (\*this).get\_array()[i][j];

}

}

for (int i = 0; i < width; i++) delete[] matrix[i];

delete[] matrix;

width = \_width;

length = \_length;

matrix = new B1 \* [width];

for (int i = 0; i < width; i++) matrix[i] = new B1[length];

for (int i = 0; i < width; i++)

{

for (int j = 0; j < length; j++)

{

matrix[i][j] = 0;

}

}

int wb = (D.get\_width() < width) ? D.get\_width() : width;

int lb = (D.get\_length() < length) ? D.get\_length() : length;

for (int i = 0; i < wb; i++)

{

for (int j = 0; j < lb; j++)

{

matrix[i][j] = D.get\_array()[i][j];

}

}

}

template <class A3>

A3\*& Matrix<A3>::operator[](const int i)

{

if ((i > width) || (i > length) || (i < 0))throw length\_error("uncorrect index");

return matrix[i];

}

template<class A1>

Matrix<A1>& Matrix<A1>::operator =(Matrix& lhs)

{

if (this == &lhs)

{

return (\*this);

}

for (int i = 0; i < width; i++) delete[] matrix[i];

delete[] matrix;

width = lhs.get\_width();

length = lhs.get\_length();

matrix = new A1 \* [width];

MemorySize(width, length);

for (int i = 0; i < width; i++)

{

for (int j = 0; j < length; j++)

{

matrix[i][j] = lhs[i][j];

}

}

return \*this;

}

template <class Type>

Matrix<Type> operator +(Matrix<Type>& lhs, Matrix<Type>& rhs)

{

if ((lhs.get\_width() != rhs.get\_width()) || (lhs.get\_length() != rhs.get\_length()))throw length\_error("matrix size not equal");

Matrix<Type> A(lhs.get\_width(), rhs.get\_length());

for (int i = 0; i < rhs.get\_width(); i++)

{

for (int j = 0; j < rhs.get\_length(); j++)

{

A[i][j] = lhs[i][j] + rhs[i][j];

}

}

return A;

}

template <class Type>

Matrix<Type> operator -(Matrix<Type>& lhs, Matrix<Type>& rhs)

{

if ((lhs.get\_width() != rhs.get\_width()) || (lhs.get\_length() != rhs.get\_length()))throw length\_error("matrix size not equal");

Matrix<Type> A(lhs);

for (int i = 0; i < rhs.get\_width(); i++)

{

for (int j = 0; j < rhs.get\_length(); j++)

{

A[i][j] = A[i][j] - rhs[i][j];

}

}

return A;

}

template <class E>

bool operator ==(Matrix<E>& lhs, Matrix<E>& rhs)

{

if ((lhs.get\_width() != rhs.get\_width()) || (lhs.get\_length() != rhs.get\_length()))

{

return false;

}

else

{

for (int i = 0; i < lhs.get\_width(); i++)

{

for (int j = 0; j < lhs.get\_length(); j++)

{

if (lhs[i][j] != rhs[i][j]) return false;

}

}

}

return true;

}

template <class F>

Matrix<F> operator \*(Matrix<F>& lhs, Matrix<F>& rhs)

{

if (lhs.get\_length() != rhs.get\_width())throw length\_error("matrix size uncorrect");

Matrix<F> C(lhs.get\_width(), rhs.get\_length());

F S;

for (int i = 0; i < lhs.get\_width(); i++)

{

for (int j = 0; j < rhs.get\_length(); j++)

{

S = 0;

for (int k = 0; k < rhs.get\_width(); k++)

{

S = S + lhs.get\_array()[i][k] \* rhs.get\_array()[k][j];

}

C[i][j] = S;

}

}

return C;

}

template <class R>

ostream& operator <<(ostream& out, Matrix<R>& rhs)

{

for (int i = 0; i < rhs.get\_width(); i++)

{

for (int j = 0; j < rhs.get\_length(); j++)

{

out << rhs.get\_array()[i][j] << " ";

}

out << endl;

}

return out;

}

template <class K>

istream& operator >> (istream& in, Matrix<K>& rhs)

{

if ((rhs.get\_length() == 0) || (rhs.get\_width() == 0))

{

cout << "Vvedite width&Length" << endl;

int a, b;

in >> a >> b;

rhs.set\_width\_length(a, b);

}

for (int i = 0; i < rhs.get\_width(); i++)

{

for (int j = 0; j < rhs.get\_length(); j++)

{

in >> rhs[i][j];

}

}

return in;

}

**«main.cpp»**

#include <iostream>

#include <cstdlib>

#include <malloc.h>

#include <time.h>

#include "matrix.h"

#include "vector.h"

#include <vector>

int main()

{

try

{

/\*Matrix<int>B;

Matrix<int>A(2, 2);

Matrix<int>C(2,2);

//--------------------------------------------------------------------------

cout << C.get\_width() << "getter shiriny\n";

cout << C.get\_length() << "getter dliny\n";

cout << C.get\_array()[0][0] << "getter matrix\n";

//--------------------------------------------------------------------------

B.set\_width\_length(1, 1);

A.set\_width\_length(1, 1);

cout << B.get\_width() <<" "<< B.get\_length() << "setter dliny i shiriny\n";

//--------------------------------------------------------------------------

//cin >> A >> B;

cout << "operator >> \n";

cout << B[0][0] << "operator [ ]\n";

//--------------------------------------------------------------------------

C = A + B;

cout << C << "operator +\n";

C = A - B;

cout << C << "operator -\n";

cout << A \* B << "operator \*\n";

cout << (A == B) << "operator ==\n";

//---------------------------------------------------------------------------

Vector<float>G(5);

Vector<float>H(5);

Vector<float>I(H);

A.set\_width\_length(3, 3);

Vector<float> L(3);

Matrix<float>s(3, 3);

//---------------------------------------------------------------------------

cout << H << "operator <<\n";

cout << H.get\_vector()[0] << "getter vector\n";

cout << H[0] << "operator [ ]\n";

G.set\_length(5);

//---------------------------------------------------------------------------

//cin >> G;

//cin >> H;

cout << H / G << "operator /\n";

I= G + H;

G.set\_length(3);

L = s \* L;

float p = G \* H;

cout << I;

Vector<int>K(10000);

In\_Vector(K);

cout << BubbleSort(K) << endl;

cout << InsertionSort(K) << endl;

cout << QuickSort(K, 0, K.get\_length() - 1) << endl;

Matrix <int> P(1, 2);

P[0][0] = 1;

cout << P[0][0];/\*/

//------------------------------------------------------------------------------

cout << "QuickSort" << endl;

vector<double> A(3);

Vector<int>Z(1000);

Matrix<int>N(1000, 1000);

int j = 0;

for (int i = 100000; j < 3; i \*= 2)

{

Z.set\_length(i);

In\_Vector(Z);

clock\_t start = clock();

QuickSort(Z, 0, Z.get\_length() - 1);

clock\_t end = clock();

A[j] = (double)(end - start) / CLOCKS\_PER\_SEC;

j++;

}

for (int i = 0; i < 3; i++)

{

cout << A[i] << endl;

}

}

catch (length\_error& a)

{

cout <<a.what() << endl;

}

return 0;

}